

Iterative improvement algorithms

In many optimization problems, **path** is irrelevant;
the goal state itself is the solution

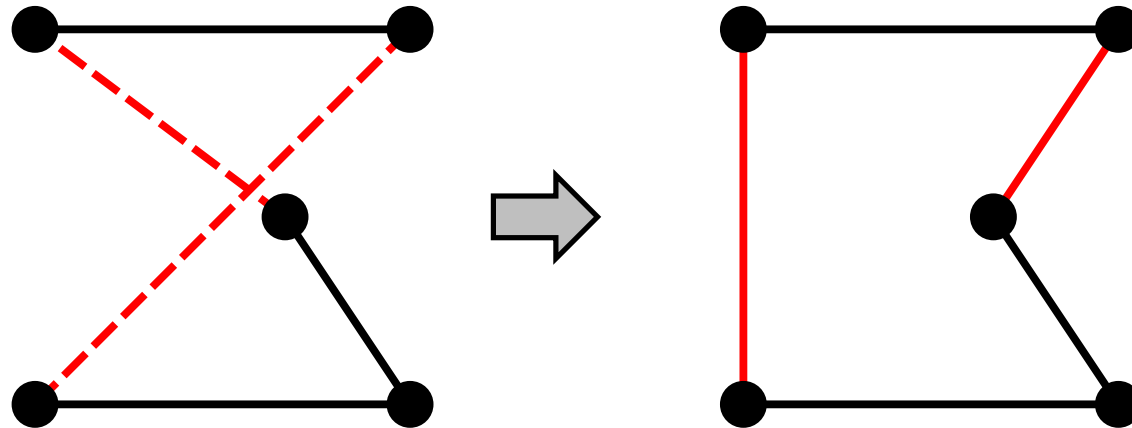
Then state space = set of “complete” configurations;
find **optimal** configuration, e.g., TSP
or, find configuration satisfying constraints, e.g., timetable

In such cases, can use **iterative improvement** algorithms;
keep a single “current” state, try to improve it

Constant space, suitable for online as well as offline search

Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

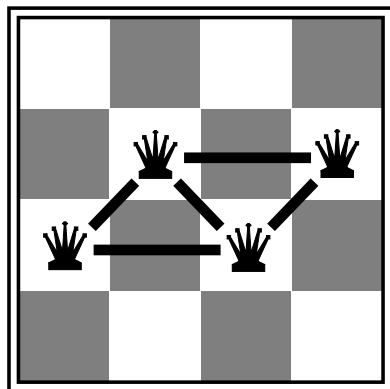


Variants of this approach get within 1% of optimal very quickly with thousands of cities

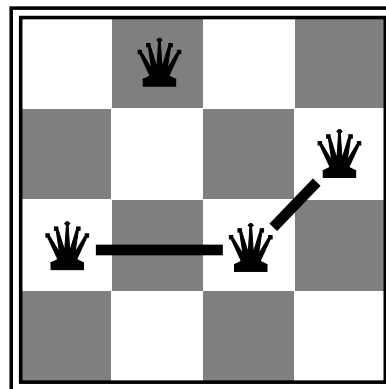
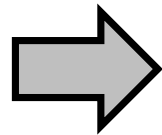
Example: n -queens

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

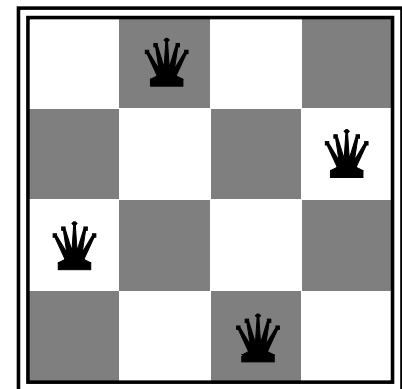
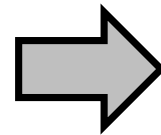
Move a queen to reduce number of conflicts



$h = 5$



$h = 2$



$h = 0$

Almost always solves n -queens problems almost instantaneously for very large n , e.g., $n = 1\text{million}$

Hill-climbing (or gradient ascent/descent)

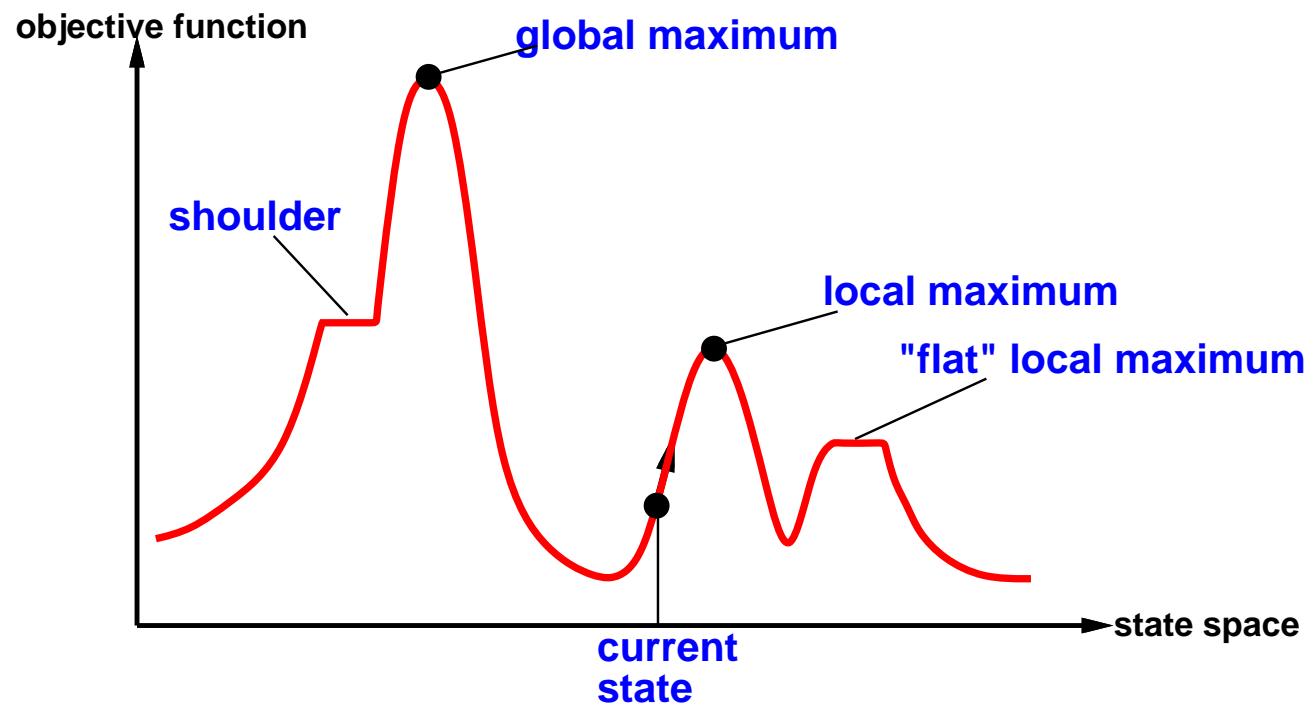
“Like climbing Everest in thick fog with amnesia”

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  inputs: problem, a problem
  local variables: current, a node
                     neighbor, a node

  current ← MAKE-NODE(INITIAL-STATE[problem])
  loop do
    neighbor ← a highest-valued successor of current
    if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
    current ← neighbor
  end
```

Hill-climbing contd.

Useful to consider state space landscape



Random-restart hill climbing overcomes local maxima—trivially complete

Random sideways moves 😊 escape from shoulders 😞 loop on flat maxima